MQ update

67

January 2005

In this issue

A tci PUBLICATION

MQ Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Colin Smith
E-mail: info@xephon.com

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs $380.00 in the USA and Canada; £255.00 in the UK; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the July 2000 issue, are available separately to subscribers for $33.75 (£22.50) each including postage.

Contributions

When Xephon is given copyright, articles published in *MQ Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

MQ Update on-line

Code from *MQ Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at www.xephon.com/mq; you will need to supply a word from the printed issue.

Generating and executing commands in batch for MQ on z/OS

Quite often you need to automate certain tasks within the MQSeries environment. Examples include starting and stopping channels, changing the BATCHSZ at certain times to provide optimal performance for different types of workload, and get/put enable/disable a local queue at certain times. Although nearly every z/OS environment has a batch scheduler installed, not everybody has an MQSeries management tool to automate these tasks. If you have a tool that can trap messages from the SYSLOG and can trigger a batch job, this method could also be used to automatically resolve error situations, for example to reset a channel sequence number if it has got out of sync.

The following batch job together with the two REXX EXECs allows you to create MQSeries commands based on wildcards that can be used as input to the CSQUTIL batch utility. This is a cheap and effective way to execute MQSeries commands at scheduled times or in specific situations, while having a report of what has happened at the same time. This is often required for audit reasons and not every MQSeries management tool records the actions it performs and provides a log. The JCL consist of four steps:

- Step 1 – delete two datasets used by the REXX EXECs. (If this job has to be run in parallel, the dataset names must be modified to allow this; for example, a time stamp within the dataset name.)
- Step 2 – MQGETOBJ EXEC. This EXEC produces a dataset containing MQSeries object names that are generated by using MQSeries DISPLAY commands. The rules for MQSeries DISPLAY commands apply. The input parameters for the EXEC are the queue manager name, the parse-string, and an MQSeries DISPLAY command. The EXEC runs the DISPLAY command against the

specified queue manager and writes the SYSPRINT output to a temporary dataset. The parse-string is then used to scan the output. To find out what has to be specified for parse-string you will need to know the output format of the DISPLAY command and check the string before the MQSeries object you want to display. If, for example, the DISPLAY command is:

```
DIS QLOCAL(IVP.*)
```

the output looks like:

```
CSQ9022I QT01 CSQMDRTS ' DISPLAY QLOCAL' NORMAL COMPLETION
 DISPLAY QLOCAL(IVP.INST.QL.TEST1)
CSQN205I   COUNT=       3, RETURN=00000000, REASON=00000000
CSQM401I QT01
 QUEUE(IVP.INST.QL.TEST1)
 TYPE(QLOCAL)
 QSGDISP(QMGR)
```

In this case the parse-string has to be set to QUEUE because the EXEC scans the SYSPRINT output for 'QUEUE('. Having found a line with this string and only this string, the string behind 'parse-string(', up to the close bracket ')', is gathered and stored in another dataset. In this example it would be *IVP.INST.QL.TEST1*.

- Step 3 – MQCRECMD EXEC. The dataset produced by Step 2 is input to this step. Also, an MQSeries command template has to be provided. This EXEC uses the template, looks for the string <OBJNAME> within the template, and replaces it with the MQSeries object names produced by the previous step. For example, to start channels the template looks like:

```
START CHANNEL(<OBJNAME>)
```

If, for instance, three channel names were produced as a result of the second step, three START CHANNEL commands would be created in this EXEC, one for each channel object. It is also possible to generate a sequence of commands by specifying a sequence of templates. For example, some alterations for channels take place only when the channel is stopped and started again. In this case you specify three templates in the order you want them to be executed. For example:

```
STOP  CHANNEL(<OBJNAME>)
RESET CHANNEL(<OBJNAME>) SEQNUM(1)
START CHANNEL(<OBJNAME>)
```

Nine MQSeries commands would be generated if three channel names were produced in Step 2. All commands are written to a sequential dataset.

- Step 4 – CSQUTIL. The output dataset from Step 3 can simply be used as input for the CSQUTIL batch utility.

MQCRECMD JCL

```
//jobname   JOB  'account',CLASS=1,MSGCLASS=S,
//           MSGLEVEL=(1,1),NOTIFY=&SYSUID
//*
//DELETE    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 DELETE   CSQ.MQ.OBJ.TEMP
 DELETE   CSQ.MQ.CMD.TEMP
 SET MAXCC = Ø
//*
//GETOBJS   EXEC PGM=IKJEFTØ1,DYNAMNBR=3Ø,REGION=4Ø96K
//SYSPROC    DD DSN=CSQ.EXEC,
//              DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//OBJNAMES  DD DSN=CSQ.MQ.OBJ.TEMP,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(1,1)),
//             DCB=(DSORG=PS,RECFM=FB,LRECL=8Ø,BLKSIZE=312Ø)
//*
//* Sample for SYSTSIN:
//* MQGETOBJ QTØ1 QUEUE DIS QLOCAL(IVP*)
//*
//SYSTSIN   DD *
 MQGETOBJ qmgr parsestring MQSeries-display-cmd
//*
//CREACMD   EXEC PGM=IKJEFTØ1
//SYSPROC    DD DSN=CSQ.EXEC,
//              DISP=SHR
//SYSTSPRT  DD SYSOUT=*
```

```
//*
//* Sample for CMDTPL:
//* ALTER QLOCAL(<OBJNAME>) GET('DISABLED')
//*
//CMDTPL     DD *
 MQSeries-cmd(<OBJNAME>) optional-parameters
//*
//OBJNAMES   DD DSN=CSQ.MQ.OBJ.TEMP,
//             DISP=(SHR,DELETE)
//CMDOUT     DD DSN=CSQ.MQ.CMD.TEMP,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(1,1)),
//             DCB=(DSORG=PS,RECFM=FB,LRECL=8Ø,BLKSIZE=312Ø)
//SYSTSIN    DD *
 MQCRECMD
//*
//RUNCMD    EXEC PGM=CSQUTIL,PARM=('QTØ1')
//SYSPRINT   DD SYSOUT=*
//SYSIN     DD *
    COMMAND DDNAME(INPUT) FAILURE(CONTINUE)
//INPUT      DD DSN=CSQ.MQ.CMD.TEMP,
//             DISP=(SHR,DELETE)
//*
```

MQCRECMD EXEC

```
/* REXX */
/*
 * This REXX reads all MQSeries object names provided by EXEC MQGETOBJ,
 * reads the command template, and creates the MQSeries commands.
 * The commands can then be used as input for CSQUTIL.
 *
 * Usage:
 *  call MQCRECMD
 *
  *==================================================================
 *
 * DDname CMDTPL has to contain the MQSeries command template. The
 * MQSeries object name (OBJNAME) must be enclosed in <>. This will
 * be substituted later with all MQSeries objects from the dataset
 * allocated with DDname OBJNAMES and, for each object name provided,
 * an MQSeries command will be generated.
 *
 * Examples:
 * START CHANNEL(<OBJNAME>)
 * STOP CHANNEL(<OBJNAME>) MODE(QUIESCE)
 * DISPLAY CHANNEL(<OBJNAME>)
 * RESET CHANNEL(<OBJNAME>) SEQNUM(1)
 * ALTER CHANNEL(<OBJNAME>) CHLTYPE(SDR) BATCHSZ(1000)
 *
 * DISPLAY QSTATUS(<OBJNAME>)
 * ALTER QLOCAL(<OBJNAME>) GET('DISABLED')
 * DELETE QLOCAL(<OBJNAME>) PURGE
 *
 * DISPLAY PROCESS(<OBJNAME>)
 *
 *==================================================================
 *
 * The following DDnames have to be provided in the JCL:
 *   OBJNAMES  Contains the MQ object names created by MQGETOBJ
 *   SYSPROC   IKJEFT01 EXEC dataset
 *   SYSTSPRT  Will contain a informational log
 *   SYSTSIN   MQCRECMD
 *   CMDTPL    MQSeries command template (see above)
 *   CMDOUT    Will contain the MQSeries commands which are to be
 *             used as input for CSQUTIL
 *             (PS, FB, 80)
 *
 *==================================================================
 */
trace off
"EXECIO *  DISKR CMDTPL  (STEM cmdtpl."
"EXECIO 0  DISKR CMDTPL  (FINIS"
"EXECIO 1  DISKR OBJNAMES"
if rc ¬= 0 then do
  say "RC = " rc
  say "Error in EXECIO for DDNAme OBJNAMES"
  exit 99
end
i=0
do while rc ¬= 2
  pull qname
  qname = strip(qname,"B" , " ")
  do  j = 1 to cmdtpl.0
     chgres   = change(cmdtpl.j,"<objname>",qname)
     chgres   = change(cmdtpl.j,"<OBJNAME>",qname)
     push chgres
     "EXECIO 1  DISKW CMDOUT"
  end
  "EXECIO 1  DISKR OBJNAMES"
  i = i + 1
end
say ""
say cmdtpl.0 * i "MQSeries command(s) generated."
say ""
"EXECIO 0  DISKR OBJNAMES (FINIS"
"EXECIO 0  DISKW CMDOUT  (FINIS"
exit 0
```

```
change: PROCEDURE
parse arg string, old, new
if old = "" then return new||string
out = ""
do while pos(old,string) ¬= Ø
  parse var string pre_part (old) string
  out = out||pre_part||new
end
return out||string
```

MQGETOBJ EXEC

```
/* REXX */
/*
 * This REXX returns all MQSeries object names matching the wildcard
 *
 * Usage:
 *  call MQGETOB  <qmgr> <parsestring> <command>
 *
 *   <qmgr>         the QueueManager to connect to
 *   <parsestring>  string to parse for the object name. Object name
 *                  is expected in a pair of parenthesis right after
 *                  the parsestring.
 *   <command>      the display command to execute
 *================================================================
 *
 * Customization:
 *
 * The line near the end:
 *
 *   "CALL 'SYS1.MQ.SCSQAUTH(CSQUTIL)'  '"qmgr"' "
 *
 * must be modified to point to the dataset containing CSQUTIL.
 *================================================================
 *
 * Examples:
 * call MQGETOB  CSQ1 CHANNEL DIS CHANNEL(CSQ1.TO.*)
 * call MQGETOB  CSQ1 QUEUE   DIS QLOCAL(IVP.*)
 * call MQGETOB  CSQ1 QUEUE   DIS QMODEL(*)
 * call MQGETOB  CSQ1 QUEUE   DIS QREMOTE(IVP*)
 * call MQGETOB  CSQ1 QUEUE   DIS QALIAS(IVP.AL.*)
 * call MQGETOB  CSQ1 PROCESS DIS PROCESS(*)
 *================================================================
 *
 * The following DDnames are used/allocated by the REXX:
 *   SYSPRINT  for the output of the queue names
 *   CSQUCMD   for the DISPLAY COMMAND
 *
 * The following DDnames have to be provided in the JCL:
```

```
 *   OBJNAMES  Will contain the retrieved object names which can
 *             be used as input for EXEC MQCRECMD
 *             (PS, FB, 8Ø)
 *   SYSPROC   IKJEFTØ1 EXEC dataset
 *   SYSTSPRT  Will contain an informational log
 *   SYSTSIN   Call MQGETOBJ with parameter string (see above)
 *===================================================================
 */
trace off
parse arg qmgr parsestring command
command     = strip(command,B," ")
qmgr        = strip(qmgr,B," ")
parsestring = strip(parsestring,B," ")
say "Parameters for MQGETOBJ utility: "
say " QueueManager:" qmgr
say " Parsestring :" parsestring
say " MQCommand   :" command
say " MQ objects retrieved:"
/* Call CSQUTIL to get objnames */
"ALLOC DD(SYSPRINT) NEW RECFM(F,B) LRECL(8Ø) BLKSIZE(312Ø) "
"ALLOC DD(CSQUCMD)  NEW RECFM(F,B) LRECL(8Ø) BLKSIZE(312Ø) "
"ALLOC DD(SYSIN)    NEW RECFM(F,B) LRECL(8Ø) BLKSIZE(312Ø) "
call csqutil qmgr command
parsestring = parsestring"("
ii = Ø
"EXECIO 1  DISKR SYSPRINT"
do while rc ¬= 2
  pull inrec
  ii = ii + 1
  if ( index(inrec,parsestring) > Ø ) then do
     /*
      * prefstr makes sure that nothing DIS CHANNEL(CSQ1.TO.*)
      * is being processed.
      */
     parsecmd1 = "parse var inrec prefstr '"parsestring"' objname ')'"
     interpret parsecmd1
     if prefstr = " "  & objname ¬= "" then do
       say "" objname
       push objname
       "EXECIO 1  DISKW OBJNAMES"
     end
  end
  "EXECIO 1  DISKR SYSPRINT "
end
"EXECIO Ø  DISKR SYSPRINT (FINIS"
"FREE  DD(SYSPRINT) "
"FREE  DD(CSQUCMD) "
"FREE  DD(SYSIN)  "
Exit(Ø)
csqutil: PROCEDURE
```

```
parse arg qmgr command
push " COMMAND "
"EXECIO 1 DISKW SYSIN   (FINIS"
push command
"EXECIO 1 DISKW CSQUCMD (FINIS"
"CALL 'SYS1.WEBS.MQ.SCSQAUTH(CSQUTIL)'  '"qmgr"' "
Return(Ø)
```

Gerald Stark
Mainframe Technical Specialist
MQ Architect (UK)

Integrating COBOL applications with Microsoft BizTalk Server 2004 – part 2

This month we conclude the article describing how to integrate COBOL applications with Microsoft BizTalk Server 2004.

HOW TO CREATE ORCHESTRATION

The following steps are to be performed:

- Get COBOL copybooks.
- Import copybooks into annotated XML schema.
- Edit XML schema.
- Generate C# code or assembly from the XML schema.
- Reference generated assembly in the BizTalk project.
- Optionally generate BizTalk XML schemas from generated assemblies.
- Create pipelines in the BizTalk project.
- Create BizTalk orchestration.
- Create a port that uses the MQ adapter and pipelines previously created.

- Compile and deploy orchestration.
- Bind ports.

Get COBOL copybooks

The cycle starts with the identification of a host application that BizTalk orchestration will communicate with. From the perspective of a developer, a host application is a set of COBOL copybooks (which represent data) and operations (which represent data flow). To get a host's application data available to a BizTalk orchestration, COBOL copybooks are retrieved.

Let's say we have a COBOL copybook called *cobtyp1.cob*:

```
       Ø1  COB-TYPES-STRUCT.
             Ø3  CNAME PIC X(8) VALUE 'DUGTYP1'.
             Ø3  F1 COMP-1 VALUE Ø.111E+2.
             Ø3  B1 COMP-2 VALUE Ø.112E+2.
             Ø3  INT1 PIC 9(9) BINARY VALUE 1ØØ.
             Ø3  INT1S PIC S9(9) BINARY VALUE -1Ø1.
             Ø3  INT2 PIC 9(4) BINARY VALUE 1Ø2.
             Ø3  INT2S PIC S9(4) BINARY VALUE -1Ø3.
             Ø3  D1 PIC 9(5) VALUE 1Ø5.
             Ø3  D1S PIC S9(5) BINARY VALUE -1Ø6.
             Ø3  DP1 PIC 9(5) COMP-3 VALUE 1Ø5.
             Ø3  DP1S PIC S9(5) COMP-3 VALUE -1Ø6.
             Ø3  DV1 PIC 9(5)V99 VALUE ZEROS.
             Ø3  DV1S PIC S9(5)V99 BINARY VALUE -1Ø8.
             Ø3  DPV1 PIC 9(5)V9(2) COMP-3 VALUE 1Ø9.
             Ø3  DPV1S PIC S9(5)V9(2) COMP-3 VALUE -11Ø.
```

Import copybooks into annotated XML schema

Retrieved COBOL copybooks are converted to an XML form (annotated XML schemas) before further processing. These schemas provide a typed XML data view, preserving information about the data layout. The *CobolCopybookParser.exe* design-time tool is used to import COBOL copybooks:

```
CobolCopybookParser.exe /o:output-path /i:input-path\cobtyp1.cob /n:
Sample\cobtyp1
```

where:

- *output-path* is the path to output the generated XML schema.
- *input-path\cobtyp1.cob* is the path to the COBOL copybook to import.
- *targetNmespace* is the target namespace for the generated XML schema.

The result produced (called *cobtyp1.xsd*) looks like:

```
<?XML version="1.Ø" encoding="utf-8"?>
<xsd:schema XMLns:xsd="http://www.w3.org/2ØØ1/XMLSchema"
  XMLns:cobol= "http://www.multiconn.com/cobol-schema/122ØØ3"
  XMLns:s="Sample\cobtyp1"
  targetNamespace="Sample\cobtyp1"
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  id="COBTYP1">

  <xsd:group name="copybook">
    <xsd:sequence>
      <xsd:element ref="s:COB-TYPES-STRUCT" />
    </xsd:sequence>
  </xsd:group>

  <xsd:element name="COB-TYPES-STRUCT" type="s:COB-TYPES-STRUCT" />

  <xsd:complexType name="COB-TYPES-STRUCT">
    <xsd:attribute default="DUGTYP1" name="CNAME" type="xsd:string">
      <xsd:annotation>
        <xsd:appinfo>
          <cobol:info type="char" length="8" />
        </xsd:appinfo>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute default="11.1" name="F1" type="xsd:float">
      <xsd:annotation>
        <xsd:appinfo>
          <cobol:info type="float" />
        </xsd:appinfo>
      </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute default="11.2" name="B1" type="xsd:double">
      <xsd:annotation>
        <xsd:appinfo>
          <cobol:info type="double" />
        </xsd:appinfo>
      </xsd:annotation>
    </xsd:attribute>
    ...
  </xsd:complexType>
</xsd:schema>
```

This document can be edited by any XML editor (XML schema editor), where XML schema types can be customized.

Edit XML schema

Annotated XML schema allow some level of customization. COBOL types can be mapped to different XML schema types. Simple types can be converted to and from enumerations. COBOL field direction (input/output) specification is available.

To view an original COBOL copybook you can insert the following XML processing instruction after the declaration:

```
<?xml ... ?>
  <?xml-stylesheet type="text/xsl" href="<schema cobol view path>"?>
```

where *<schema cobol view path>* stands for *<install path>\Design-time tools\Annotated schema view\schema cobol view.xslt*, and *<install path>* is the Multiconn BizTalk Extensibility installation path.

You can view a COBOL copybook if you open this XML schema in a browser.

Generate C# code or assembly from XML schema

At this stage we generate C# code from the annotated XML schema. The *Cobol Importer.exe* design-time tool is used to generate C# code:

```
Cobol Importer.exe" /n:code-namespace o:output-path input-
path\cobtyp1.xsd
```

where:

- *code-namespace* is the namespace of the generated code.
- *output-path* is the path to output the generated code.

- *input-path\cobtyp1.xsd* is the path to an annotated XML schema from which to generate the code.

The code generated is a regular .NET class annotated with XMLXxx attributes for XML serialization, and with CobolLayout attributes for COBOL serialization.

To create assembly from the generated code you can either create a C# project, which includes this code, or specify the **/ assembly:** option in the *Cobol Importer.exe* tool.

Reference generated assembly in the BizTalk project

To use generated assemblies in the BizTalk project, they need to be referenced. Right-click a BizTalk project in solution explorer and select the **Add Reference** option.

Types from generated assemblies are used in pipeline designer to specify the data format in the CICS DPL serializer and deserializer components. The messages based on these types may also be used in BizTalk orchestration in order to form data that is passed to a port.

Optionally generate BizTalk XML schemas from generated assemblies

Generated assemblies can be used to create BizTalk XML schemas. Right-click a BizTalk project in solution explorer and select options **Add** and then **Add Generated Items**, select the **Generate Schemas** template and click **open**. Select **Assembly schema generator** in the **Document type** field, and specify the assembly path in the **Input file** field. Click **OK** to generate the BizTalk XML schemas from the assembly.

Use these XML schemas in BizTalk orchestration to create messages. Note that messages based on a generated .NET type and messages based on XML schema derived from this type are compatible; however, only XML schema-based messages can be used in the BizTalk map files.

Create pipelines in the BizTalk project

To communicate with a host application through the CICS DPL bridge, you must create a port that uses CICS DPL pipelines, which prepare raw data for the host application. You need to create both a **Send pipeline** and **Receive pipeline**.

Drop the **CICS DPL Serializer** pipeline component from the **Toolbox** at the **Assemble** stage of the **Send pipeline**, and the **CICS DPL Deserializer** pipeline component at the **Disassemble** stage of the **Receive pipeline**.

Note: if you do not see CICS DPL pipeline components on the **BizTalk pipeline components** toolbox, right-click the toolbox and select **Add/Remove Items**. In the dialog opened, select the **BizTalk pipeline components** tab, and then check **CICS DPL Serializer** and **CICS DPL Deserializer**. Press **OK**, and these components will appear in the toolbox.

ContentEncoding property is the name of the encoding that the CICS DPL pipeline components use to convert text data. Default encoding is **IBM500**.

Documents collection allows users to specify a set of message types that pipeline component can process. Items in this collection have the DocumentSpec property, which allows the selection of the available types that support COBOL layout serialization. An item's ContentLength property optionally allows you to specify the raw data length. The ApplicationName property is used to specify the DPL program name.

Create BizTalk orchestration

Create a BizTalk orchestration that implements your custom logic. You can use the messages and variables based on the types generated using design-time tools. Create a new **Send-Receive** port on the port surface of the orchestration. This port will be configured later. It will use the MQ adapter and pipelines created earlier. Use this port to send requests and receive responses from the host application.

Create a port that uses the MQ adapter and pipelines previously created

In order to create and configure a port that uses MQ adapter do the following:

- Drag and drop the **Port** component from the BizTalk components ToolBox onto the orchestration. The **Port Configuration Wizard** appears.
- Provide a name for the port.
- Select the port type.
- Select a communication pattern.

  Note: a port that uses MQ adapter can use both the **One-Way** and **Request-Response** communication patterns.
- Select the direction of communication and select either **Specify later** or **Direct** for the port binding.
- Complete the wizard.

When the port is created, you can specify a message type of **Operation Message**.

Compile and deploy orchestration

To bind ports, you need to compile and deploy the BizTalk project.

Bind ports

After deployment you have to bind the orchestration's ports to the real send/receive ports.

To create and bind a send port, do the following in BizTalk Explorer:

1 Right-click **Send Ports** and create a new port with the communicating pattern corresponding to the port in the orchestration.

2 Rename it, if you wish.

3 Set **Transport Type** to MQ.

4 Select the **Address** property and then click the '...' button.

5 Specify **Queue manager**, **Input queue**, and **Output queue** properties if you are using MQSeries server and all other properties if you are using MQSeries client.

6 Click **OK**.

7 Click the **Send** node from the left-hand side.

8 Set the property to **Send Pipeline** for the name of your send CICS DPL pipeline.

9 For the solicit-response port, set the property to **Receive Pipeline** for the name of your receive CICS DPL pipeline.

10 Click **OK**.

11 Select *<server-name>.BizTalkMgmtDb.dbo* database and click the **Refresh** button on the BizTalk Explorer toolbar.

12 Expand the **Orchestrations** node. You will see your orchestrations there.

13 Double-click on it and bind the orchestration's send port to the newly-created port.

To create and bind a receive port, do the following in BizTalk Explorer:

1 Right-click **Receive Ports** and create a new port with the communicating pattern corresponding to the port in the orchestration.

2 Rename it, if you wish.

3 Click **OK**.

4 For a request-response port, set the property of the name of your send CICS DPL pipeline to **Send Pipeline**.

5 Right-click **Receive Locations** under the newly-created receive port, and click **Add Receive Location…**.

6 Set **Transport Type** to MQ.

7 Select the **Address** property and then click the '...' button.

8 Specify **Queue manager**, **Input queue**, and **Output queue** properties if you are using MQSeries server and all other properties if you are using MQSeries client.

9 Click **OK**.

10 Select a **Receive Handler** from the drop-down list.

11 Set the property of the name of your receive CICS DPL pipeline to **Receive Pipeline**.

12 Click **OK**.

13 Select *<server-name>.BizTalkMgmtDb.dbo* database and click the **Refresh** button on the BizTalk Explorer toolbar.

14 Expand the **Orchestrations** node. You will see your orchestrations there.

15 Double-click on it and bind the orchestration's send port to the newly-created port.

Now you can start your orchestration.

CONCLUSION

The integration of legacy applications with Microsoft BizTalk Server 2004 broadens the horizons for both sides. For legacy software this approach allows data to be obtained from diverse sources. For BizTalk and its clients this approach gives direct access to the reliable services provided by legacy applications.

Arthur and Vladimir Nesterovsky (Israel) © Xephon 2005

Display tool for WebSphere MQ objects on Unix

For WebSphere MQ (WMQ) administrators on Unix systems it is quite easy to view and modify WMQ objects using the command line interface **runmqsc** or have a look at the contents of queues using the IBM sample programs like **amqsbcg**. It is also possible to use graphical user interfaces such as MQMON (the IBM SupportPac MO71) for this task. In general, a graphical interface is easier to use because users are guided to the functions they need, but these tools mostly need a running command server and a client channel definition, which may be undesirable for security reasons.

Although WMQ administration tools are quite usable for administrators, they are very difficult to use by business people who want to have just a quick look at certain objects and contents of queues. In this case you need a tool that provides business people with the information they want, without the need to train them on the WMQ administration tools. Even when they are interested in technical details, they usually will not be able to spend enough time using these tools to become familiar with getting the information they want.

THE WMQ DISPLAY TOOL

Intention for a WMQ display tool

In the past, some of my customers wanted to have a quick look at some WMQ objects and queue contents to see what was going wrong when their applications fail. The tools described above are quite useful for WMQ administrators, who use them several times a day, but they may be unsuitable for business people just wanting to have a brief look at WMQ objects only once or twice per week. To provide these people with an easy-to-use WMQ viewing tool, I wrote the script described in this article. It is possible to configure this script in a secure way, without opening any back doors, like WMQ client connections would do.

How the WMQ display tool works

The WMQ display tool is designed for Unix systems and is developed as a Korn shell script. This script uses a configuration file, which defines queues and channel to display or browse. These objects are listed during the execution of the script and the user may choose a queue or channel by entering a number shown in front of the object. Business users now need to know only the name of the queue or channel to have a look at it (which, in fact, is often difficult enough for business people).

The script is designed as a substitute for the login shell. It has to be used in the file */etc/passwd* as the defined user shell:

```
mqusr1:x:12345:1234::/home/mqusr1:/home/mqusr1/mqview
mqusr2:x:23451:1234::/home/mqusr2:/home/mqusr2/mqview
mqusr3:x:34512:1234::/home/mqusr3:/home/mqusr3/mqview
```

The script itself captures control sequences such as *Ctrl-C* by issuing the command **trap**:

```
trap "exec /bin/ksh -e $HOME/mqview" HUP INT ERR TERM
```

This command captures the signals HUP, INT, ERR, and TERM, and runs the command in between the single quotes instead (the script itself). If something goes wrong (eg the path name in the script is wrong), the user will be logged off. There is now a chance for a user to get a shell environment. On some Unix derivatives the script must be defined as a valid login shell before it can be used in */etc/passwd*.

The users who will be enabled to look at the WMQ objects have to be members of the same group. This group needs permissions to the objects that they want to look at. The permissions may be set using **setmqaut**. It is also possible to add the users to the group mqm instead. Because these users cannot get a normal shell environment, this avoids a security hole.

Using the WMQ display tool

The WMQ display tool is very easy to use. The users just have to log in (using telnet, rsh, ssh, or whatever) and follow the script instructions. In the first (main) menu the user is asked to select a function. The following functions are available:

```
Select the script function:
q)      display WebSphere MQ message queues
qc)     display WebSphere MQ cluster queues
qs)     display WebSphere MQ queue status
b)      browse WebSphere MQ queues
c)      display WebSphere MQ channel
cs)     display WebSphere MQ channel status

e)      exit

Select an option
```

When the user, for example, selects the letter *q*, he will get the following menu:

```
The following queues may be displayed:

1)      SYSTEM.DEFAULT.LOCAL.QUEUE
2)      SYSTEM.DEFAULT.ALIAS.QUEUE
3)      SYSTEM.DEFAULT.REMOTE.QUEUE
4)      SYSTEM.DEFAULT.MODEL.QUEUE
5)      SYSTEM.DEAD.LETTER.QUEUE

q)      quit

Enter a queue number
```

The queue list displayed above is defined in the configuration file (lines 23 to 28 in the sample configuration file below). Now the user has to enter the number of the queue he wants to have a look at. The script sets up a simple **runmqsc** command and executes it:

```
echo "display queue(SYSTEM.DEFAULT.LOCAL.QUEUE)" | runmqsc TESTQM
5724-B41 (C) Copyright IBM Corp. 1994, 2002.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager TESTQM.


     1 : display queue(SYSTEM.DEFAULT.LOCAL.QUEUE)
AMQ8409: Display Queue details.
   DESCR(WebSphere MQ Default Local Queue)
   PROCESS( )                              BOQNAME( )
   INITQ( )                                TRIGDATA( )
   CLUSTER( )                              CLUSNL( )
   QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)       CRDATE(2004-05-28)
   CRTIME(11.26.54)                        ALTDATE(2004-05-28)
   ALTTIME(11.26.54)                       GET(ENABLED)
   PUT(ENABLED)                            DEFPRTY(0)
   DEFPSIST(NO)                            MAXDEPTH(5000)
   MAXMSGL(4194304)                        BOTHRESH(0)
   SHARE                                   DEFSOPT(SHARED)
   HARDENBO                                MSGDLVSQ(PRIORITY)
   RETINTVL(999999999)                     USAGE(NORMAL)
   NOTRIGGER                               TRIGTYPE(FIRST)
   TRIGDPTH(1)                             TRIGMPRI(0)
   QDEPTHHI(80)                            QDEPTHLO(20)
   QDPMAXEV(ENABLED)                       QDPHIEV(DISABLED)
   QDPLOEV(DISABLED)                       QSVCINT(999999999)
   QSVCIEV(NONE)                           DISTL(NO)
   DEFTYPE(PREDEFINED)                     TYPE(QLOCAL)
   SCOPE(QMGR)                             DEFBIND(OPEN)
   IPPROCS(0)                              OPPROCS(0)
   CURDEPTH(0)
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
Press <RETURN> to continue
```

The user now has to press the *Return* key to return to the selection menu.

SAMPLE CONFIGURATION

Example of a configuration file

This sample enables business users to have a look at some default queues and channels (lines 23 to 28 and lines 56 to 61) and one cluster queue (lines 34 and 35). The users will also be able to browse the queues defined in lines 49 and 50. A queue status is possible for local queues only (line 41 to 43). These objects are visible only on the queue manager, TESTQM, which is defined at the beginning of the configuration file (line 17).

```
 1 ##############################################################
 2 #
 3 #   Configuration file for the script 'mqview'. This file
 4 #   contains all system specific data and is used by
 5 #   'mqview'.
 6 #
 7 #   October, 12th 2004
 8 #   Hubert Kleinmanns
```

```
#   Senior Consultant
#
###################################################################

#
# Define the name of the queue manager.
#

LOCAL_QMGR="TESTQM"

#
# Define a list of queues, which may be displayed.
#

MQSERIES_QUEUES="\
   SYSTEM.DEFAULT.LOCAL.QUEUE \
   SYSTEM.DEFAULT.ALIAS.QUEUE \
   SYSTEM.DEFAULT.REMOTE.QUEUE \
   SYSTEM.DEFAULT.MODEL.QUEUE \
   SYSTEM.DEAD.LETTER.QUEUE"

#
# Define a list of cluster queues, which may be displayed.
#

MQSERIES_CL_QUEUES="\
   MY.CLUSTER.QUEUE"

#
# Define a list of local queues, to display queue status.
#

MQSERIES_LOCAL_QUEUES="\
   SYSTEM.DEFAULT.LOCAL.QUEUE \
   SYSTEM.DEAD.LETTER.QUEUE"

#
# Define a list of local queues, which may be browsed.
#

MQSERIES_BROWSE_QUEUES="\
   SYSTEM.DEAD.LETTER.QUEUE"

#
# Define a list of channel, which may be displayed.
#

MQSERIES_CHANNEL="\
   SYSTEM.DEF.SDR \
   SYSTEM.DEF.RCVR \
   SYSTEM.DEF.CLUSSDR \
   SYSTEM.DEF.CLUSRCVR \
   SYSTEM.DEF.SVRCONN"
```

Example of a setmqaut script

When the read-only users get permissions by using the program setmqaut, the lines below have to be executed by a WMQ administrator. The permissions are set for all users who are members of the group mqview. Instead of defining several individual users, it is also possible to define one single functional user, who is shared by all the business people. Last but not least, it is also possible to add these users to the group mqm (without opening a back door). Which solution has to be used depends on the security policies of your company.

```
#!/bin/ksh

# Include the configuration file.
. $HOME/mqview.config

# Set the 'read-only' group.
GROUP=mqview

# First allow access (connect and display) to the queue manager.
setmqaut -m $LOCAL_QMGR -t qmgr -g $GROUP +connect +dsp

# Now allow displaying of queue attributes.
for q in $MQSERIES_QUEUES
do
   setmqaut -m $LOCAL_QMGR -t q -n $q -g $GROUP +dsp
done

# Now allow browsing of queues.
for q in $MQSERIES_LOCAL_QUEUES
do
   setmqaut -m $LOCAL_QMGR -t q -n $q -g $GROUP +browse
done
```

DESCRIPTION OF THE CODE

The script code contains an endless loop (line 147 to 273), which sets up the main menu. Then the script calls the function mq_function (lines 58 to 139) and executes the **runmqsc** commands.

Main loop

First the parameter *COMMAND* defines the action that has to be performed by the script. If this parameter is set to *browse queue* (line 203), the script will run the IBM sample program **amqbcg**. In all other cases the parameter *COMMAND* contains a valid **runmqsc** command (eg *display queue* in line 158). The parameters *OBJ_NAME* and *OBJ_NAMES* contain strings which are needed to set up the output messages in the selection menu. These parameters have no other function.

The WMQ objects that have to be displayed or browsed are stored as the array *OBJ_ARR* (eg in line 163). The first member of this array is always the unused name DUMMY. This is to start the numbering of the WMQ names with 1 instead of 0. In the next line, the script stores the number of WMQ objects in the parameter *OBJ_CNT*.

Function mq_function

The function mq_function first displays a selection menu. It first prints a headline, then it loops over the members of the array *OBJ_ARR* (except the first dummy entry) and displays a line for each object, beginning with a number followed by the name of the WMQ object.

Next the script asks for an input value. The user may enter a number of a WMQ object or *q* to quit and return to the main menu. If the user enters a valid number, the **runmqsc** command is executed, otherwise an error message is shown.

Possible extensions to the script

It is quite easy to extend this script by adding some non-WMQ functionality. In fact, in my customers' environments there were some file browsing functions and the use of the **top** utility implemented. I removed these functions from the sample because this is an MQ journal. But it is quite easy to add such functions or others like database requests to this script. But be careful, do not implement tools that allow the calling of shell commands, such as **vi** or **more**! When you, for example, want to implement a file-browsing tool, try the freeware tool **less** instead of **more**. Re-compile **less** with the *SECURE* option to disable shell commands.

LISTING OF THE WMQ DISPLAY TOOL

```
#!/bin/ksh
#
######################################################################
#
#   Script for enabling restricted access to a system.
#   This script is used as a shell and prevents the user
#   from executing programs that are not explicitly allowed.
#   Some additional steps must be performed, to make
#   the restricted user secure:
#
#   Entry in '/etc/group', create a new group:
#      mqview::1234:
#
#   Entry in '/etc/passwd', create one or more users as
#   member of this group and give them this script instead
#   of a shell:
#      mqusr1:x:12345:1234:MQ display user:/home/mqusr1:/home/mqusr1/mqview
#      mqusr2:x:23451:1234:MQ display user:/home/mqusr2:/home/mqusr2/mqview
#      mqusr3:x:34512:1234:MQ display user:/home/mqusr3:/home/mqusr3/mqview
#
#   Copy this script and the configuration file to
#   the home directory of the users.
#
#   October, 12th 2004
#   Hubert Kleinmanns
#   Senior Consultant
#
######################################################################

# The script is called again when one of the signals
# (eg CTRL-C) is received.
trap "exec /bin/ksh -e $HOME/mqview" HUP INT ERR TERM

PATH="/usr/bin:/opt/mqm/samp/bin"

######################################################################
#
#   Include the configuration file, which contains the
#   system-specific attributes.
#
##################################################################

. $HOME/mqview.config

##################################################################
#
#   The function below executes the required WebSphere
#   MQ commands. Which objects have to be displayed are
#   controlled by the parameters above. Nothing has to
#   be modified below.
#
##################################################################

#
# Function to display WebSphere MQ parameters.
#

function mq_function
{
   # Clear the screen.
   clear

   # Display the menu head line.
   echo ""
   if [ "$COMMAND" = "browse queue" ]
   then
      echo "The contents of the following queues may be shown:"
   else
      echo "The following $OBJ_NAMES may be displayed:"
   fi
   echo ""

   # Display the menu entries.
   let NUM=1
   while [ $NUM -le $OBJ_CNT ]
   do
      echo "$NUM)      ${OBJ_ARR[$NUM]}"

      let NUM+=1
   done

   # Display the menu trailer.
   echo ""
   echo "q)      quit"
   echo ""
   echo "Enter a $OBJ_NAME number"

   # Read the input value.
   read NUM

   # Leave the menu with 'q'.
   if [ "$NUM" = "q" ]
   then
      FUNC=""
   # Check now the input.
   else
      # Now check whether the input number is valid.
      if [ "$NUM" -ge 1 -a "$NUM" -le $OBJ_CNT ]
      then
         # Clear screen again.
         clear

         # Browse a queue.
         if [ "$COMMAND" = "browse queue" ]
         then
  # Perform the browse command and display the output to the screen.
            CMD="amqsbcg ${OBJ_ARR[$NUM]}"
            eval $CMD
         # Perform a MQSC command.
         else
            type=""

   # Make a dummy put on cluster queues (to be sure, it is visible).
            if [ "$OBJ_NAME" = "cluster queue" ]
            then
               amqsput ${OBJ_ARR[$NUM]} > /dev/null 2>&1 <<EOF

EOF
            elif [ "$COMMAND" = "display qstatus" ]
            then
               type=" type(handle)"
            fi

    # Perform the MQSC command and display the output to the screen.
CMD="echo \"$COMMAND(${OBJ_ARR[$NUM]})$type\" | runmqsc $LOCAL_QMGR"
            echo $CMD
            eval $CMD
         fi
      # Display an error message otherwise.
      else
         echo ""
         echo "You entered an invalid number!"
      fi

      # Hold the function, to have a look at the output.
      echo "Press <RETURN> to continue"
      read ANSWER
   fi
```

```
}

#
# Main loop of the MQ administration wrapper (endless loop).
#

FUNC=""

while [ 1 ]
do
   case $FUNC in
      # Leave the program with 'e'.
      e)
         exit 0
         ;;

      # Display queues.
      q)
         # Set up the 'display queue' parameters.
         COMMAND="display queue"
         OBJ_NAME="queue"
         OBJ_NAMES="queues"

         # Set up the MQ objects to display.
         set -A OBJ_ARR DUMMY $MQSERIES_QUEUES
         OBJ_CNT='echo $MQSERIES_QUEUES | wc -w'

         # Call now the function.
         mq_function
         ;;

      # Display cluster queues.
      qc)
         # Set up the 'display cluster queue' parameters.
         COMMAND="display qcluster"
         OBJ_NAME="cluster queue"
         OBJ_NAMES="cluster queues"

         # Set up the MQ objects to display.
         set -A OBJ_ARR DUMMY $MQSERIES_CL_QUEUES
         OBJ_CNT='echo $MQSERIES_CL_QUEUES | wc -w'

         # Call now the function.
         mq_function
         ;;

      # Display queue status.
      qs)
         # Set up the 'display queue status' parameters.
         COMMAND="display qstatus"
```

```
          OBJ_NAME="queue status"
          OBJ_NAMES="queue states"

          # Set up the MQ objects to display.
          set -A OBJ_ARR DUMMY $MQSERIES_LOCAL_QUEUES
          OBJ_CNT='echo $MQSERIES_LOCAL_QUEUES | wc -w'

          # Call now the function.
          mq_function
          ;;

       # Browse queues.
       b)
          # Set up the 'browse queue' parameters.
          COMMAND="browse queue"
          OBJ_NAME="queue"

          # Set up the MQ queues to browse.
          set -A OBJ_ARR DUMMY $MQSERIES_BROWSE_QUEUES
          OBJ_CNT='echo $MQSERIES_BROWSE_QUEUES | wc -w'

          # Call now the function.
          mq_function
          ;;

       # Display channel.
       c)
          # Set up the 'display channel' parameters.
          COMMAND="display channel"
          OBJ_NAME="channel"
          OBJ_NAMES="channel"

          # Set up the MQ objects to display.
          set -A OBJ_ARR DUMMY $MQSERIES_CHANNEL
          OBJ_CNT='echo $MQSERIES_CHANNEL | wc -w'

          # Call now the function.
          mq_function
          ;;

       # Display channel status.
       cs)
          # Set up the 'display channel status' parameters.
          COMMAND="display chstatus"
          OBJ_NAME="channel"
          OBJ_NAMES="channel"

          # Set up the MQ objects to display.
          set -A OBJ_ARR DUMMY $MQSERIES_CHANNEL
          OBJ_CNT='echo $MQSERIES_CHANNEL | wc -w'
```

```

         # Call now the function.
         mq_function
         ;;

      # Print the main menu.
      *)
         # Clear screen.
         clear

         # Display the head line of the main menu.
         echo ""
         echo "Select the script function:"
         echo ""

         # Display the entries of the main menu.
         echo "q)      display WebSphere MQ queues"
         echo "qc)      display WebSphere MQ cluster queues"
         echo "qs)      display WebSphere MQ queue status"
         echo "b)      browse WebSphere MQ queues"
         echo "c)      display WebSphere MQ channel"
         echo "cs)      display WebSphere MQ channel status"

         # Display the trailer of the main menu.
         echo ""
         echo "e)      exit"
         echo ""
         echo "Select an option"

         # Read the function.
         read FUNC
         FUNC="'echo $FUNC | tr '[:upper:]' '[:lower:]''"
         ;;
   esac
done
```

Hubert Kleinmanns
Senior Consultant
N-Tuition Business Solutions AG (Germany)

Introduction to, and usage of, the WebSphere MQ JMS Admin tool

This article is ideal for those with some previous working experience and knowledge of using WebSphere MQ, and some knowledge of the concepts of Sun's Java Message Service (JMS) spec, WebSphere Application Server, and the Java Naming and Directory Interface (JNDI).

For more information about the Sun Java Message Service spec and API visit http://java.sun.com/products/jms/overview.html.

Note: the scope of the article covers distributed platforms like Windows 2000/XP, Linux, HP, AIX, and Solaris. The snapshots shown are specific to Windows. It does not take into account z/OS and OS/390 specifications for the JMS Admin tool, which may differ.

INTRODUCTION

WebSphere MQ provides an administration tool called JMS Admin. This tool is primarily used to configure WebSphere MQ as a JMS. The tool can be used to create and define the properties of WebSphere MQ-administered JMS objects, examples of which will be seen later on.

The JMS Admin tool has been around since the release of the MA88 by IBM, a SupportPac that provides support for developing WebSphere MQ applications in Java, and also since the release of WebSphere MQ Version 5.2. It was designed also to work with the IBM WebSphere Application Server Version 3.5, Version 4.0 of JNDI, and also with other JNDI providers.

USING THE JMS ADMIN TOOL

What is supported

Administrators can create up to eight types of WebSphere MQ JMS object. These are listed below with their respective keywords:

1 MQConnectionFactory * – CF

2 MQQueueConnectionFactory – QCF

3 MQTopicConnectionFactory – TCF

4 MQQueue – Q

5 MQTopic – T

6 MQXAConnectionFactory * – XACF

7 MQXAQueueConnectionFactory – XAQCF

8 MQXATopicConnectionFactory – XATCF

9 JMSWrapXAQueueConnectionFactory** – WSQCF

10 JMSWrapXATopicConnectionFactory** – WSTCF

* These objects can be administered only with JMS 1.1 specification.

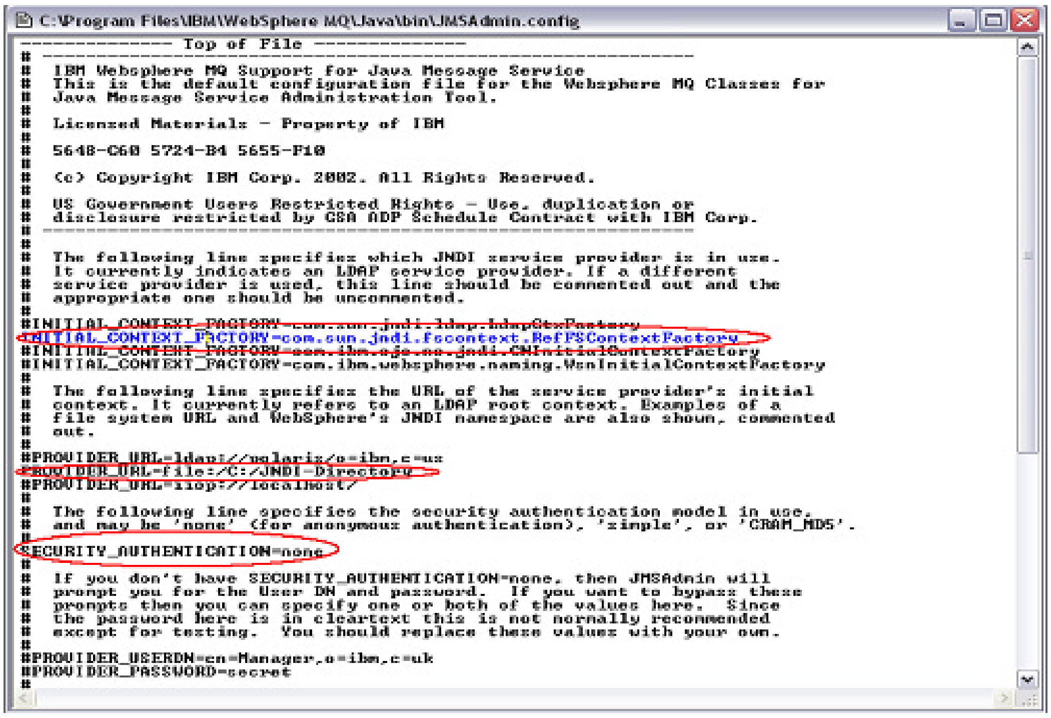
**These objects can be used only with a version of WebSphere Application Server earlier than Version 5.

How to configure the tool for use

Any of the JMS objects listed above, once defined and created, are stored within a JNDI namespace. You can configure what type of JNDI service provider to use by editing the *JMSAdmin.config* file in your WebSphere MQ root installation directory under *Java\bin* (see Figure 1).

The *INITIAL_CONTEXT_FACTORY* section provides four types of JNDI services provider. To use a particular type, uncomment that line by removing the # from the beginning of the line. In Figure 1, the *com.sun.jndi.fscontext.RefFSContextFactory* JNDI provider will be used to store the JMS object definitions once they are created. This means that the file system will be used to store the objects.

Next, to use your selected JNDI service provider, you have to select the root of that provider's Initial Context by selecting one of the three options in the *PROVIDER_URL* section. The



C:\Program Files\IBM\WebSphere MQ\Java\bin\JMSAdmin.config

```
-------------- Top of File --------------
# ----------------------------------------------------------------------
#  IBM Websphere MQ Support for Java Message Service
#  This is the default configuration file for the Websphere MQ Classes for
#  Java Message Service Administration Tool.
#
#  Licensed Materials - Property of IBM
#
#  5648-C60 5724-B4 5655-F10
#
#  (c) Copyright IBM Corp. 2002. All Rights Reserved.
#
#  US Government Users Restricted Rights - Use, duplication or
#  disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# ----------------------------------------------------------------------
#
#  The following line specifies which JNDI service provider is in use.
#  It currently indicates an LDAP service provider. If a different
#  service provider is used, this line should be commented out and the
#  appropriate one should be uncommented.
#
#INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
#
#  The following line specifies the URL of the service provider's initial
#  context. It currently refers to an LDAP root context. Examples of a
#  file system URL and WebSphere's JNDI namespace are also shown, commented
#  out.
#
#PROVIDER_URL=ldap://polaris/o=ibm,c=us
PROVIDER_URL=file:/C:/JNDI-Directory
#PROVIDER_URL=iiop://localhost/
#
#  The following line specifies the security authentication model in use,
#  and may be 'none' (for anonymous authentication), 'simple', or 'CRAM_MD5'.
#
SECURITY_AUTHENTICATION=none
#
#  If you don't have SECURITY_AUTHENTICATION=none, then JMSAdmin will
#  prompt you for the User DN and password.  If you want to bypass these
#  prompts then you can specify one or both of the values here.  Since
#  the password here is in cleartext this is not normally recommended
#  except for testing.  You should replace these values with your own.
#
#PROVIDER_USERDN=cn=Manager,o=ibm,c=uk
#PROVIDER_PASSWORD=secret
#
```

Figure 1: The JMSAdmin.config file

options are LDAP, FILE, and IIOP.

For example in Figure 1, the user has selected the FILE context with the value of */C:/JNDI-Directory*. This means that all the naming and directory operations will point to this directory on the file system. This option can be used only with the initial context factory *com.sun.jndi.fscontext.RefFSContextFactory*.

Similarly, to use an LDAP as the root of the Initial Context, specify the location of the LDAP server or IIOP to access a WebSphere Application Server namespace.

Basically when the JMS Admin tool is started, it starts a session and that session needs an initial context. This initial context will act as the root of all JNDI operations carried out by the tool.

The tool also provides a security authentication feature. This feature is available and can be used only when an LDAP service provider is used. There is a *Security_Authentication* property in the JMSAdmin.config (see Figure 1) file, which can take one of three values:

- None (default)
- Simple
- CRAM-MD5.

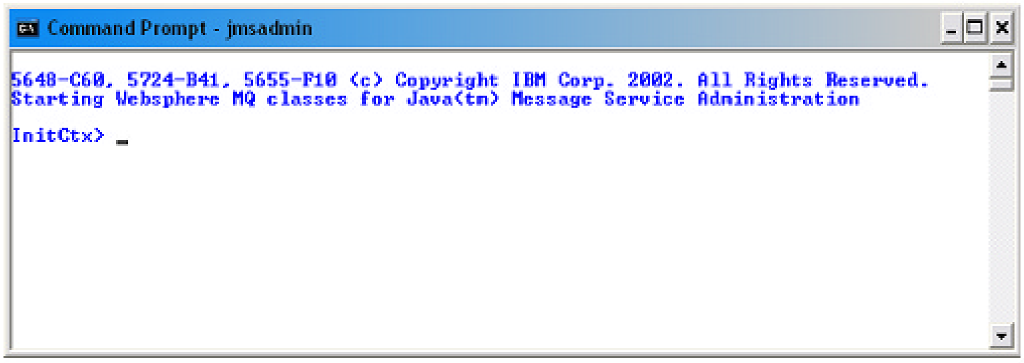
If no value is specified then *none* is used as the default.

How to create the supported JMS objects

First start the JMSAdmin tool by running the *JMSAdmin.bat* file provided in your WebSphere MQ root installation directory under *Java\bin*. If running on a Windows platform, you should see a command prompt for the Initial Context displayed (see Figure 2).

The tool accepts commands called 'Administration Commands'. The commands are also called verbs and eight types of verb are available:

1 ALTER – change at least one property of an administered object.



```
5648-C60, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2002. All Rights Reserved.
Starting Websphere MQ classes for Java(tm) Message Service Administration

InitCtx> _
```

Figure 2: The Initial Context prompt

2 DEFINE – create and store an administered object.

3 DISPLAY – display properties of one or more stored administered object.

4 DELETE – remove one or more administered object.

5 CHANGE – change the current context.

6 COPY – make a copy of a stored administered object.

7 MOVE – alter the name of a stored administered object.

8 END – close the administration tool.

As mentioned earlier, you can create objects that will be stored in a JNDI namespace using the following command syntax:

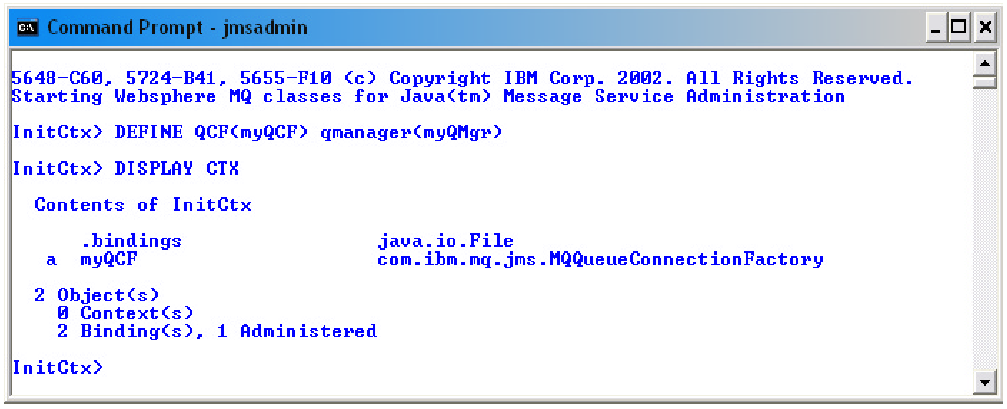
```
DEFINE TYPE(  name  ) [property]
```

For example:

```
DEFINE  QCF(myQCF) qmanager(myQMgr)
```

In the above example, *DEFINE* is a verb and *QCF* is the keyword for an MQQueueConnectionfactory JMS object.

Once objects are created, as shown in the previous step, we can display the contents of the context using the **DISPLAY**



```
5648-C60, 5724-B41, 5655-F10 (c) Copyright IBM Corp. 2002. All Rights Reserved.
Starting Websphere MQ classes for Java(tm) Message Service Administration

InitCtx> DEFINE QCF(myQCF) qmanager(myQMgr)

InitCtx> DISPLAY CTX

  Contents of InitCtx

      .bindings                  java.io.File
   a  myQCF                      com.ibm.mq.jms.MQQueueConnectionFactory

  2 Object(s)
    0 Context(s)
    2 Binding(s), 1 Administered

InitCtx>
```

Figure 3: DISPLAY CTX command

CTX command (see Figure 3).

After creating a QCF, when we display the contents of the initial context we can see the details of what is in the context. Figure 3 shows that there are two objects in context, one of which has been created by the QCF.

The *.bindings* object is the bindings file that stores the objects created. This is because the *com.sun.jndi.fscontext.RefFSContextFactory* and */C:/JNDI-Directory* settings were selected in the *JMSAdmin.config* file to use the file system for JNDI storage.

The *.bindings* file can be located in the directory specified in the provider URL and in this case it is in */C:/JNDI-Directory*. If this file is deleted, then all objects created will be lost.

To end the JMSAdmin session simply use the END verb at the *InitCtx*> command prompt.

There are a plethora of properties available to configure for the supported MQ JMS objects. Which are used also depends on what kind of JMS application is written to interact with them.

Covering all the properties is beyond the scope of this article, but a more detailed and complete guidance on using the JMSAdmin tool can be found in Chapter 5 of the *WebSphere MQ Using Java* manual. This manual is available for free download at www.elink.ibmlink.ibm.com/public/applications/publications/cgibinpbi.cgi?CTY=US&FNC=SRX&PBL=SC34606602.

Security titbit: how to use the WebSphere MQ JMSAdmin tool with client-side security with WebSphere Application Server V4

It is possible to use the JMSAdmin application to provide client-side security with WebSphere Application Server V4 with minor modifications to the *JMSAdmin.bat* script file shown below to specify the use of a sas.client.props file:

```
*************************************************************************
@echo off
```

```
rem ----------------------------------------------
rem  IBM MQSeries JMS Admin Tool Execution Script
rem  for Windows NT
rem
rem  Note that the properties passed to the java
rem  program are defaults, and should be edited
rem  to suit your installation if necessary
rem ----------------------------------------------
SET CLIENTSAS=-Dcom.ibm.CORBA.ConfigURL=file:/C:/mqjava/sas.client.props
Echo Using: %CLIENTSAS%
java %CLIENTSAS% -DMQJMS_LOG_DIR="%MQ_JAVA_INSTALL_PATH%"\log -
DMQJMS_TRACE_DIR="%MQ_JAVA_INSTALL_PATH%"\trace -
DMQJMS_INSTALL_PATH="%MQ_JAVA_INSTALL_PATH%"
com.ibm.mq.jms.admin.JMSAdmin %1 %2 %3 %4 %5
***********************************************************************
```

The security can be provided by editing the properties file named in the *JMSAdmin.bat* (in this case */C:/mqjava/ sas.client.props*) to either prompt the user for a valid username and password combination or embed the username and password within the properties file. For example, to prompt the user, the *sas.client.props* file should be edited to specify a value:

```
com.ibm.CORBA.loginSource=stdin
```

When the JMSAdmin application executes, the user will be prompted for a valid password as follows:

```
***********************************************************************
C:\mqjava>JMSAdmin -cfg jmsadmin3.config
Using : -Dcom.ibm.CORBA.ConfigURL=file:/C:/mqjava/sas.client.props

5648-C60 (c) Copyright IBM Corp. 2002. All Rights Reserved.
Starting MQSeries classes for Java(tm) Message Service Administration

InitCtx> DISPLAY CTX

  Contents of InitCtxRealm/Cell Name: myCellName
User Identity: myUserid
User Password: ********         //   a password that is not displayed


  [D] oskararg                 javax.naming.Context
   a  T                        com.ibm.mq.jms.MQQueue
  [D] domain                   javax.naming.Context
  [D] jta                      javax.naming.Context
  [D] ejsadmin                 javax.naming.Context
```

```
   a  T1                        com.ibm.mq.jms.MQQueue
  [D] jdbc                      javax.naming.Context
      SecurityCurrent
com.ibm.ejs.security.util.SecurityCurrentRef
      IncBean
com.ibm.websphere.examples.Inc._IncHome_Stub
  [D] jndi                      javax.naming.Context

  10 Object(s)
    6 Context(s)
    4 Binding(s), 2 Administered

InitCtx> END

Stopping MQSeries classes for Java(tm) Message Service Administration
************************************************************************
```

If the user wants to embed the username and password (myUserid and myPassword) in the properties file they can use:

```
com.ibm.CORBA.loginSource=properties
com.ibm.CORBA.loginUserid=myUserid
com.ibm.CORBA.loginPassword=myPassword
```

The userid/password combination is now passed automatically without user intervention.

Rohit Bhasin
Software Engineer, WebSphere MQI GUI Test Team
IBM Hursley (UK)

How to migrate Plug-In node from WMQI to WBIMB

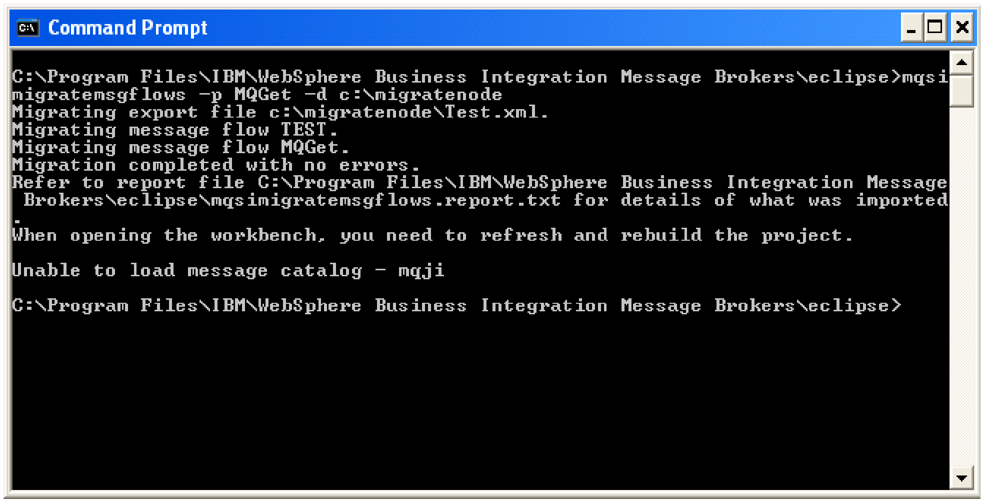
One of the limitations of message flow design is that it can process messages from only one input queue specified in the MQInput node. When the requirement is to read a 'trigger' message from one queue and then read the data message from another queue, you will need the MQGet SupportPac IA09 from WMQI. Unfortunately, at the time I was dealing with this design requirement, IA09 was not refreshed on the

SupportPac Web site (http://www-306.ibm.com/software/integration/support/supportpacs/category.html#cat1) for use in WBIMB. I could not even find it. So I had to migrate the node myself.

Plug-In node written for the previous releases should work under WBIMB because the API has not changed significantly, only the representation of the node changes from Control Center to the Eclipse plug-ins.

Step 1: create a simple message flow in Version 2.1 that consists of the node. This is just a trick since the migration utility provided works at the message flow level. The message flow doesn't even need to be working.

Step 2: detach the file and save it in a folder, for example,



```
C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse>mqsi
migratemsgflows -p MQGet -d c:\migratenode
Migrating export file c:\migratenode\Test.xml.
Migrating message flow TEST.
Migrating message flow MQGet.
Migration completed with no errors.
Refer to report file C:\Program Files\IBM\WebSphere Business Integration Message
 Brokers\eclipse\mqsimigratemsgflows.report.txt for details of what was imported
.

When opening the workbench, you need to refresh and rebuild the project.

Unable to load message catalog - mqji

C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse>
```

Figure 1: mqsimigratemsgflows command

c:\migratenode.

Step 3: shut down the WBIMB toolkit.

Step 4: run the **mqsimigratemsgflows** command as follows:

```
C:\Program Files\IBM\WebSphere Business Integration Message
Brokers\eclipse>mqsi
```

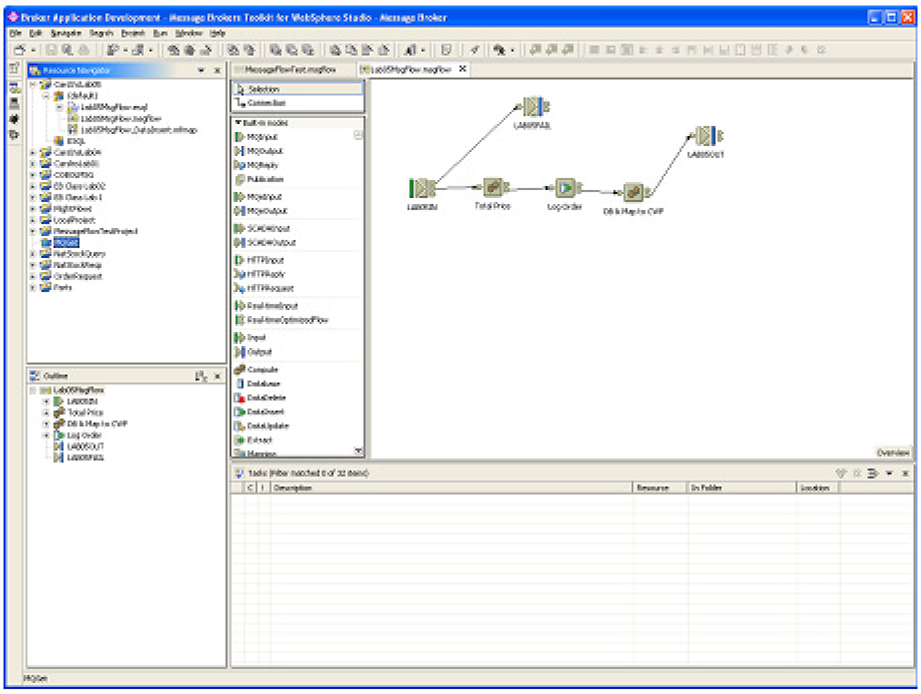

Broker Application Development - Message Brokers Toolkit for WebSphere Studio - Message Broker
Resource Navigator
Outline
LAB05IN
Total Price
Log Order
DB & Map to CVP
LAB05OUT
LAB05FAIL
MQGet

Figure 2: Resource Navigator in WBIMB toolkit

```
migratemsgflows -p MQGet -d c:\migratenode
Migrating export file c:\migratenode\Test.xml.
```

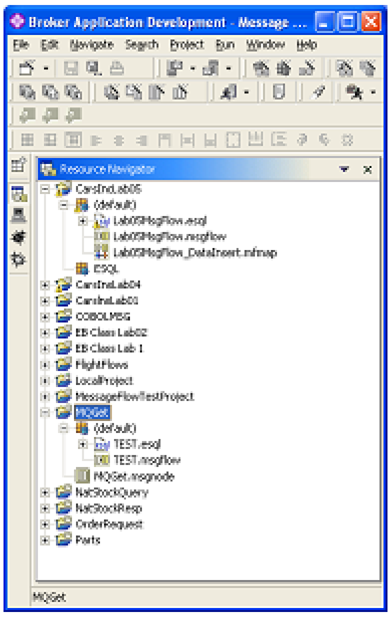


Figure 3: Project

```
Migrating message flow TEST.
Migrating message flow MQGet.
Migration completed with no errors.
```

Refer to the report file *C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse\mqsimigratemsgflows.report.txt* for details of what was imported – see Figure 1.

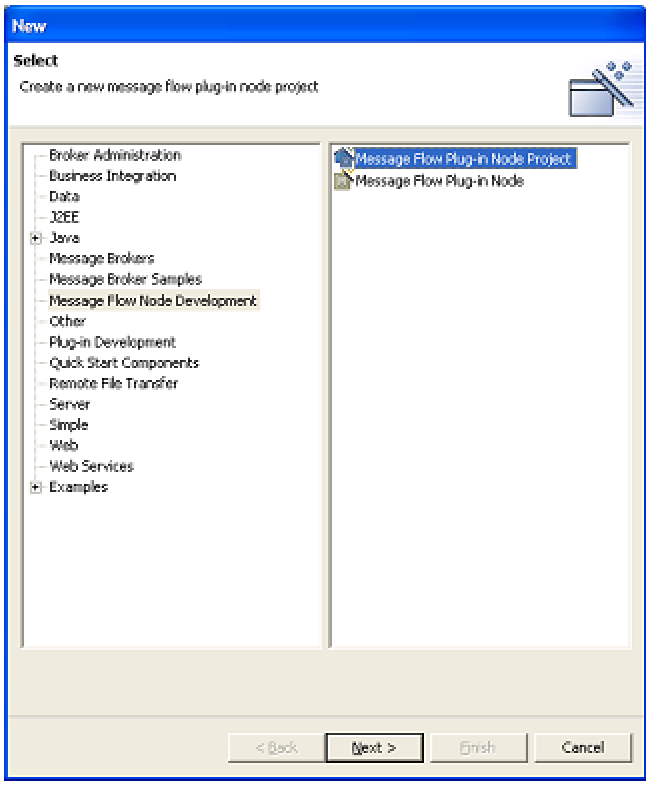
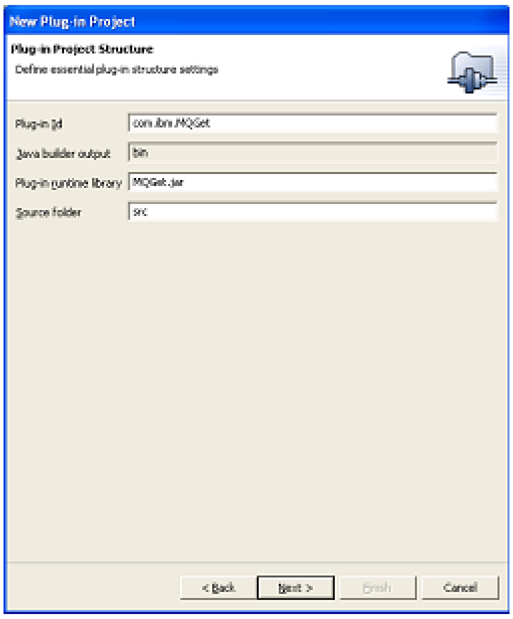


Figure 4: New project

When opening the workbench, you need to refresh and rebuild the project.

Step 5: start the WBIMB toolkit.

Step 6: the MQGet project will appear in the Resource



New Plug-in Project

Plug-in Project Structure

Define essential plug-in structure settings

Plug-in Id: com.ibm.MQSet

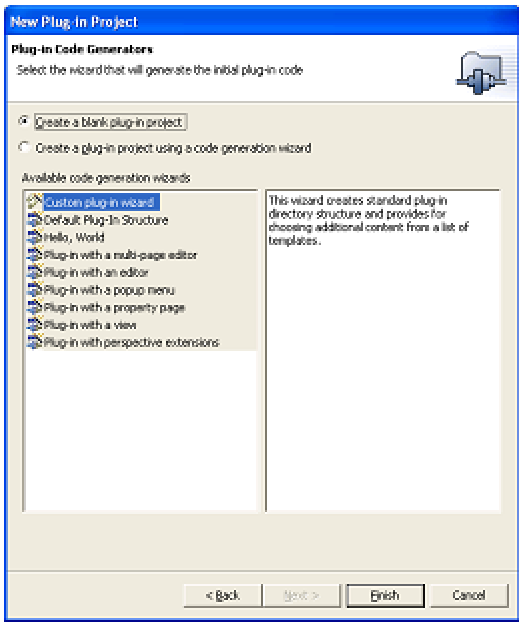
Java builder output: bin

Plug-in runtime library: MQGet.jar

Source folder: src

Figure 5: Project names

Navigator. Open the project by right-clicking on **Open Project** – see Figure 2.



New Plug-in Project

Plug-in Code Generators

Select the wizard that will generate the initial plug-in code

[✓ Create a blank plug-in project]

Create a plug-in project using a code generation wizard

Available code generation wizards

- Custom plug-in wizard
- Default Plug-In Structure
- Hello, World
- Plug-in with a multi-page editor
- Plug-in with an editor
- Plug-in with a popup menu
- Plug-in with a property page
- Plug-in with a view
- Plug-in with perspective extensions

This wizard creates standard plug-in directory structure and provides for choosing additional content from a list of templates.

Figure 6: Create a blank plug-in project

Step 7: the MQGet project will looks like Figure 3.

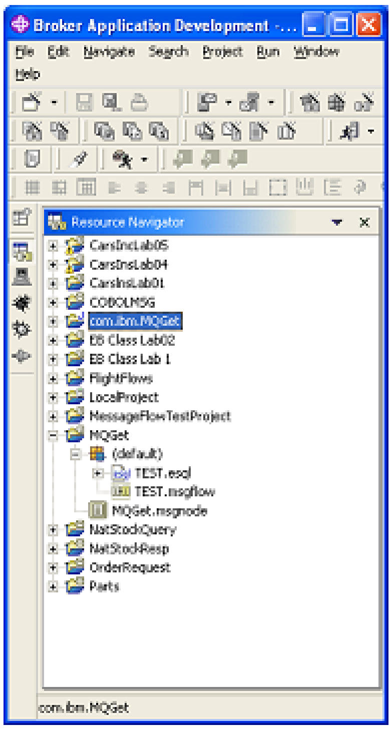


Figure 7: Available projects

The following steps create a new Plug-In node project for our migrated MQGet node.

Step 8: on the WBIMB toolkit, choose **File/New/Other**.

Step 9: in the **New Project** dialog choose M**essage Flow Node Development** and then **Message Flow Plug-in Node Project** – see Figure 4.

Step 10: click the **Next** button; enter **Migrated Nodes** in the **Category name** (this name will appear after the migration on your toolkit).

Step 11: click **Next**; enter **project names** as **com.ibm.MQGet**, leave the use default checked – see Figure 5.

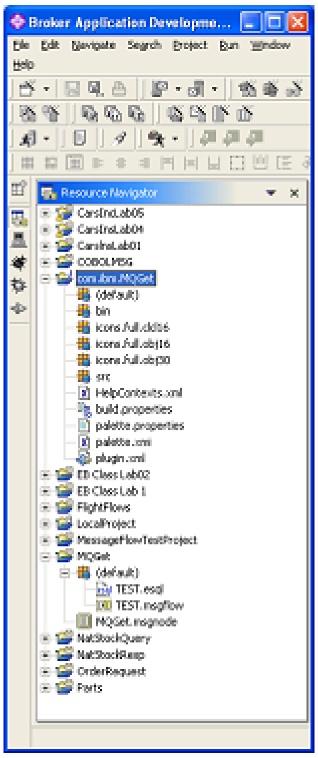


Figure 8: The com.ibm.MQGet project

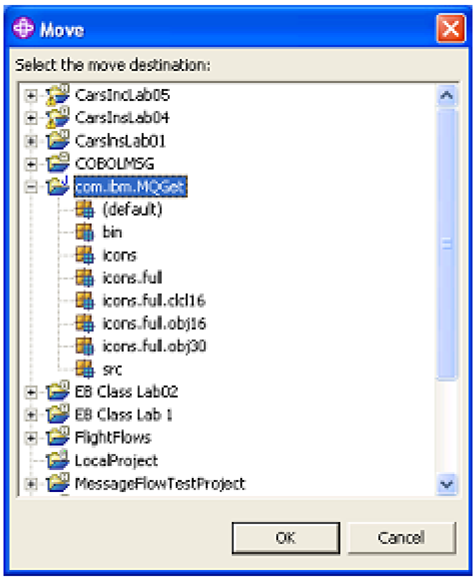
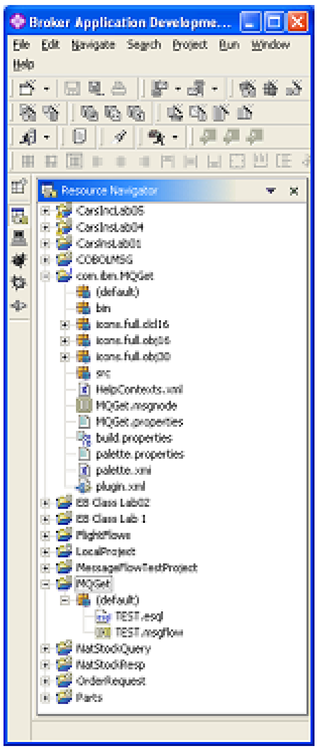
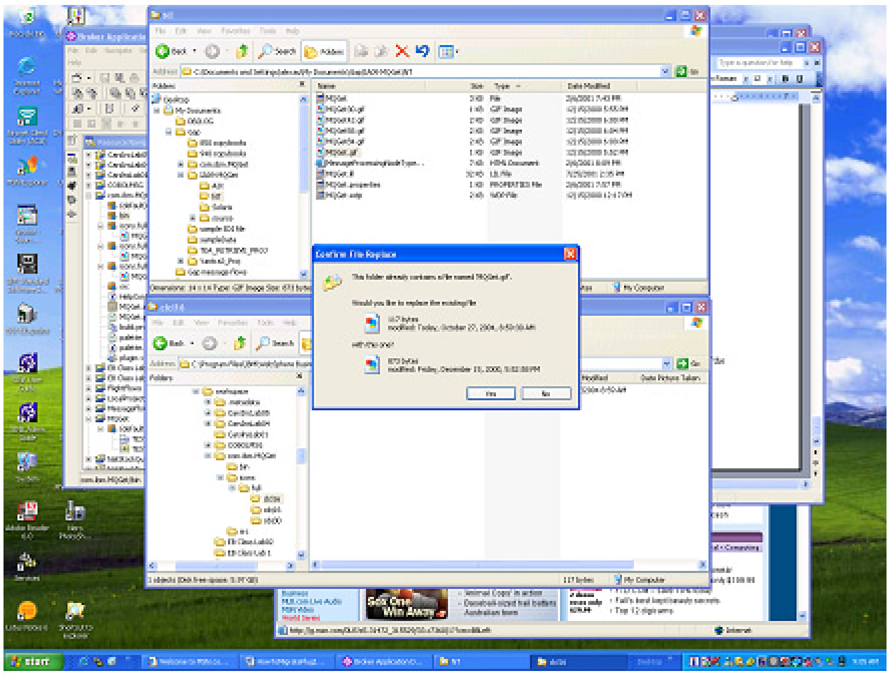


Figure 9: Move dialog box


Broker Application Developme...
File Edit Navigate Search Project Run Window Help
Resource Navigator

```
+ CarsIncLab05
+ CarsInsLab04
+ CarsInsLab01
+ COBOLMSG
- com.ibm.MQGet
    (default)
    bin
  + icons.full.clcl16
  + icons.full.obj16
  + icons.full.obj30
    src
    HelpContexts.xml
    MQGet.msgnode
    MQGet.properties
    build.properties
    palette.properties
    palette.xml
    plugin.xml
+ EB Class Lab02
+ EB Class Lab 1
+ FlightFlows
+ LocalProject
+ MessageFlowTestProject
- MQGet
  - (default)
      TEST.esql
      TEST.msgflow
+ NatStockQuery
+ NatStockResp
+ OrderRequest
+ Parts
```

Figure 10: Projects



This folder already contains a file named MQGet.gif.

Would you like to replace the existing file

with this one?

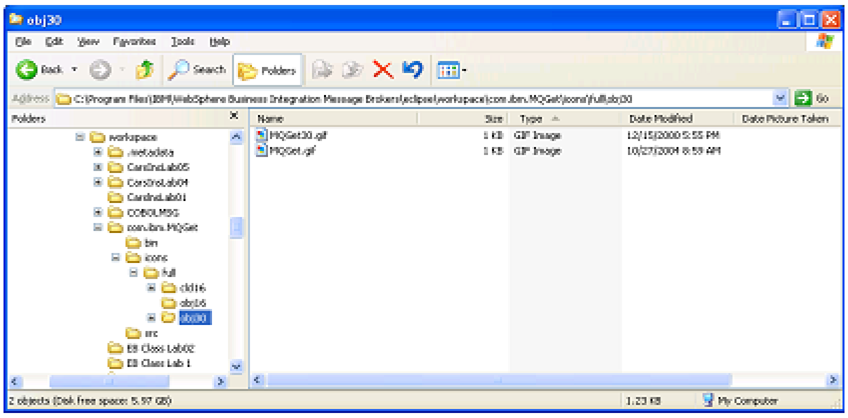
Figure 11: Replace GIFs

Step 12: click **Next**, and check the **Create a blank plug-in project** – see Figure 6.

Step 13: click **Finish**; the project will then appear in the Resource Navigator as shown in Figure 7.

Step 14: open up the com.ibm.MQGet project – see Figure 8,

Step 15: move the *MQGet.msgnode* from the MQGet project



C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse\workspace\com.ibm.MQGet\icons\full\obj30

Name	Size	Type	Date Modified	Date Picture Taken
MQGet30.gif	1 KB	GIF Image	12/15/2000 5:55 PM	
MQGet.gif	1 KB	GIF Image	10/27/2004 8:59 AM	

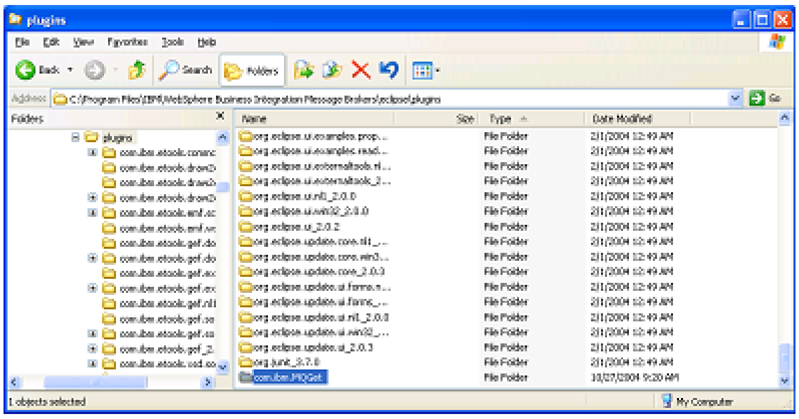
2 objects (Disk free space: 5.97 GB) 1.23 KB

Figure 12: GIF location

into the com.ibm.MQGet by right-clicking on the MQGet.msgnode and select **move**. In the move dialog box, choose the com.ibm.MQGet project, as shown in Figure 9.

Step 16: click **OK**, the resource navigator should look like Figure 10.

Step 17: next we need to move the tailored gif files from v2 to replace the vanilla gif files of Version 5. Open up the NT files folder form version v2 (in my case the path is *C:\Documents and Settings\alexau\My Documents\Gap\IA09 MQGet\NT*) and open the icon folder of Version 5 (the path is *C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse\workspace\com.ibm.MQGet\icons\full\clcl16*). Replace the *MQGet.gif* of Version 5 in the folders *clc16* and *obj16*, with the MQGet.gif file from Version 2 – see Figure 11.



plugins

Address: C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse\plugins

Name	Size	Type	Date Modified
org.eclipse.ui.examples.prop...		File Folder	2/1/2004 12:49 AM
org.eclipse.ui.examples.read...		File Folder	2/1/2004 12:49 AM
org.eclipse.ui.externaltools.nl...		File Folder	2/1/2004 12:49 AM
org.eclipse.ui.externaltools_2...		File Folder	2/1/2004 12:49 AM
org.eclipse.ui.nl1_2.0.0		File Folder	2/1/2004 12:49 AM
org.eclipse.ui.win32_2.0.0		File Folder	2/1/2004 12:49 AM
org.eclipse.ui_2.0.2		File Folder	2/1/2004 12:49 AM
org.eclipse.update.core.nl1_...		File Folder	2/1/2004 12:49 AM
org.eclipse.update.core.win3...		File Folder	2/1/2004 12:49 AM
org.eclipse.update.core_2.0.3		File Folder	2/1/2004 12:49 AM
org.eclipse.update.ui.forms.n...		File Folder	2/1/2004 12:49 AM
org.eclipse.update.ui.forms_...		File Folder	2/1/2004 12:49 AM
org.eclipse.update.ui.nl1_2.0.0		File Folder	2/1/2004 12:49 AM
org.eclipse.update.ui.win32_...		File Folder	2/1/2004 12:49 AM
org.eclipse.update.ui_2.0.3		File Folder	2/1/2004 12:49 AM
org.junit_3.7.0		File Folder	2/1/2004 12:49 AM
com.ibm.MQGet		File Folder	10/27/2004 9:20 AM

1 objects selected

Figure 13: Copy folder

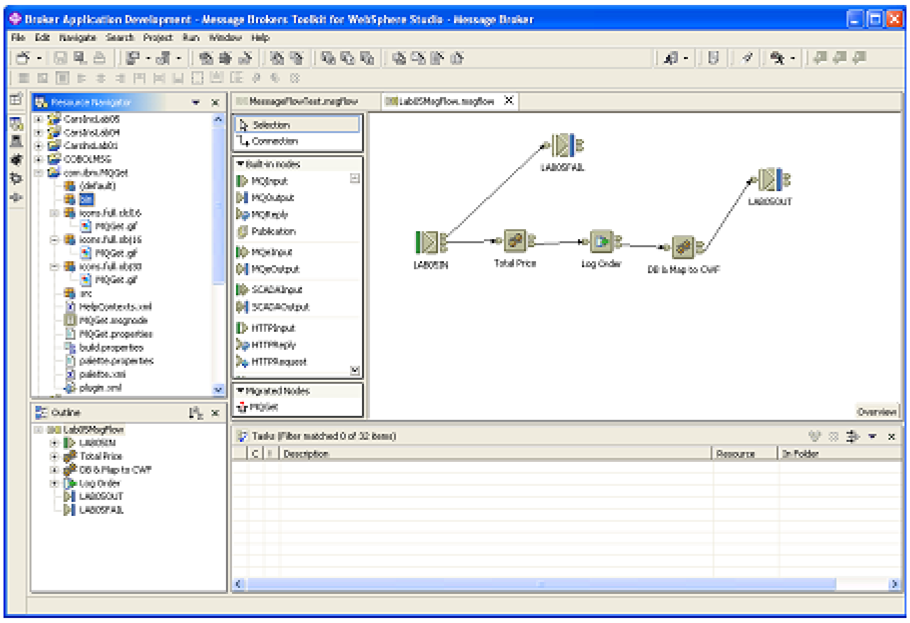


Figure 14: Node available

Step 17: also replace the MQGet.gif in the *obj30* folder (path is *C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse\workspace\com.ibm.MQGet\icons\full\obj30*) with the MQGet30.gif – see Figure 12.

Step 18: rename it to MQGet.gif after the move.

Step 19: copy the MQGet.lil from Version 2 to the bin lib (path is *C:\Program Files\IBM\WebSphere Business Integration Message Brokers\bin*) of Version 5.

Step 20: close the toolkit.

Step 21: copy the folder com.ibm.MQGet from *C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse\workspace* into the plug-in folder of eclipse, *C:\Program Files\IBM\WebSphere Business Integration Message Brokers\eclipse\plugins* – see Figure 13.

Step 22: restart the Configmgr and Broker services to refresh the MQGet.lil file.

Step 23: start the toolkit; note the Migrated Node – you should now have the MQGet node available for use – see Figure 14.

Step 24: you can delete the project com.ibm.MQGet and the project MQGet, they are no longer needed.

Alex Au
IT Architect
IBM Global Services (USA)

ILOG has announced a new connector that will enable its ILOG JRules (part of its BRMS product line) to add business rule management functionality to Business Process Management (BPM) solutions built on WebSphere Business Integration workflow software. The resulting solution will allow business users to make policy changes within a BPM application with minimal IT involvement for faster business response times to customer demands, regulations, and competitive threats.

The new connector provides seamless integration between ILOG JRules and WebSphere Business Integration software, including WebSphere MQ Workflow, allowing users to simply link their business rules with workflow tasks into the MQ Workflow Buildtime tool.

For further information contact:
ILOG, 1080 Linda Vista Ave, Mountain View, CA 94043, USA.
Tel: (650) 567 8000.
URL: www.ilog.com/corporate/releases/us/041109_ibm.cfm.

* * *

Compuware has announced the extension of its Vantage application service management product to include two new tools that help assure predictable service levels before a newly-developed project is rolled out. The tools are predictive analysis and full performance assessment.

The Vantage product interoperates with the company's QACenter tools and displays results from both on one console during performance assessment. The products run on and measure numerous platforms, including WebSphere and WebSphere MQ, SAP, Peoplesoft, Citrix, DB2, Oracle, BEA, Microsoft .Net, J2EE applications, and some systems from HP.

For further information contact:
Compuware, One Campus Martius, Detroit, MI 48226, USA.
Tel: (313) 227 7300.
URL: www.compuware.com/products/vantage/default.htm.

* * *

Active Reasoning has announced the availability of new IT compliance auditing capabilities for its Compliance and Change Management solutions.

The solutions provide companies with an audit trail by monitoring and reporting internal changes to critical IT components such as database and messaging applications that support financial flows and transactions.

Active Reasoning's IT compliance capabilities now report database schema changes to Oracle, DB2, and Microsoft SQL Server databases as well as detect internal configuration changes to WebSphere's messaging server.

For further information contact:
Active Reasoning, 1005 Elwell Court, Palo Alto, CA 94303, USA.
Tel: (650) 404 9900.
URL: www.activereasoning.com.

* * *